НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав: студент 3 курсу

ФПМ групи КВ-82

Курилко Максим Ігорович

Перевірив: Павловський В.І.

Київ – 2020

**Метою роботи** є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Загальне завдання** роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з 2-х та більше сутностей одночасно: для числових атрибутів – у рамок діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамок діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання контролер).

**Деталізоване завдання**:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти:
   1. контроль при введенні - валідація даних;
   2. перехоплення помилок **(try...except)** від сервера PostgreSQL при виконанні відповідної команди SQL.

Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. З боку батьківської таблиці необхідно контролювати **вилучення (ON DELETE)** рядків за умови наявності даних у підлеглій таблиці. З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних. Унеможливити виведення програмою на екрані системних помилок PostgreSQL шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.

1. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не програмою**, а відповідним **SQL-запитом**! Кількість даних для генерування має вводити користувач з клавіатури.
2. Для реалізації багатокритеріального пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Після виведення даних вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.
3. Програмний код організувати згідно шаблону Model-ViewController (MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). Модель, подання (представлення) та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

**Модель бази даних**

**Концептуальна модель «Онлайн кінотеатру»(Рис.1)**

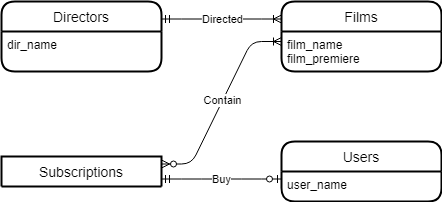
****

Рис.1 - Концептуальна модель «Онлайн кінотеатру»

**Структура БД «Онлайн кінотеатр»(Рис.2)**

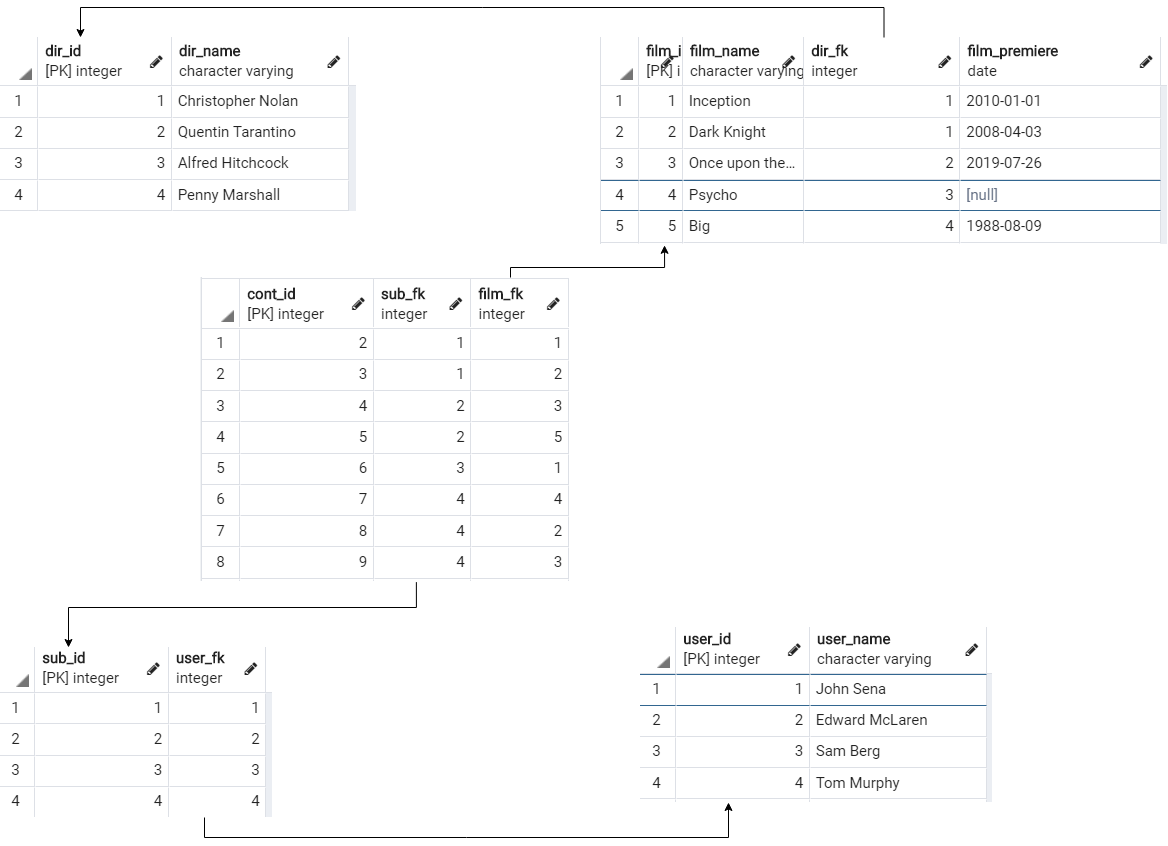
****

Рис.2 -Структура БД «Онлайн кінотеатр»

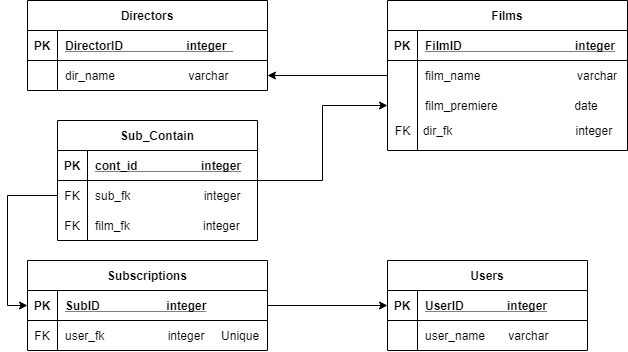
**Логічна модель БД «Онлайн кінотеатр»(Рис.3)**

Рис.3 - Логічна модель БД «Онлайн кінотеатр»

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувалась мова програмування С# та Visual Studio Code 2019. Для підключення до сервера бази даних PostgreSQL використовувався бібліотека Npgsql. Для цього створений метод, який з’єднується з базою даних.

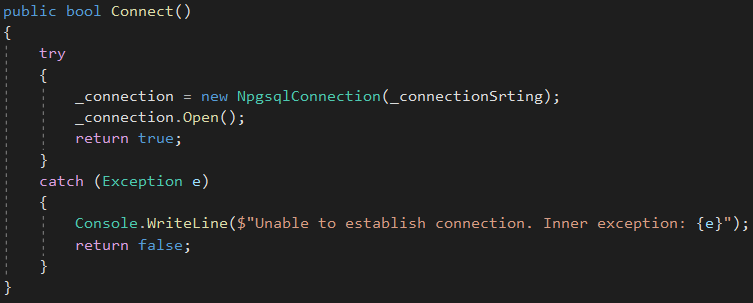


Рис.4 - Процедура під’єднання до PostgreSQL

**Опис структури програми**

Програма містить 6 модулів

* *Program* – клас, який виконує функцію Controller, в якому знаходиться функція main, що забезпечує виконання програми.
* *Model* – клас який формує запити і використовує абстрактну функцію *request*(..) для їх виконання.
* *View* – описує клас, який керує діалогом з користувачем використовуючи екземпляр класу model
* *QueryResult* – описує клас, який спадковує клас *model* і реалізовує під’єднання і виконання запитів саме з *PostgreSQL*.
* *Table*
* *ViewIO*

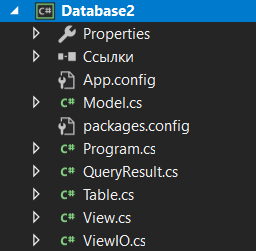


Рис.5 – Список модулів програми

**Структура меню програми**

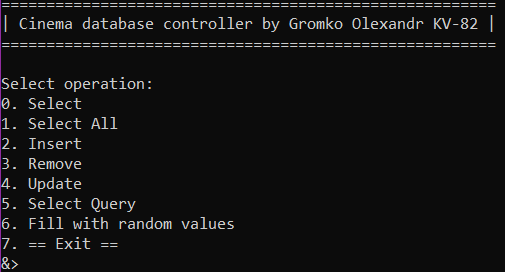


Рис.6 - Меню вводу операції

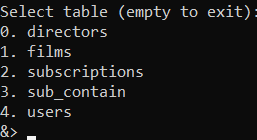


Рис.7 - Меню вводу таблиці

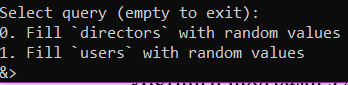


Рис.8 – Меню вибору полей для випадкового встановлення

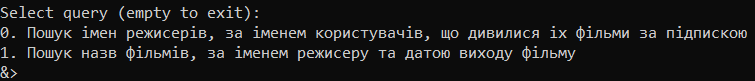


Рис.9 – Меню вибору одного з заготовлених запитів.

**Посилання для навігації**

1. Лістинги програми з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив
2. Функції для внесення
3. Приклад роботи
4. Функції для оновленяя
5. Приклад роботи
6. Результат роботи, використовуючи оператор виведення всіх полів
7. Функції для вилучення
8. Приклад виклику
9. Результат роботи
10. Лістинги програми з директивами внесення рандомізованих даних і виконання динамічних запитів у базі даних та результати виконання цих директив
11. Рандомізоване внесення даних до таблиці «Users»
12. Приклад використання
13. Виконання динамічних запитів бази даних
14. Перший запит
15. Другий запит
16. Третій запит
17. Обробка виняткових ситуацій (помилок) при введенні/вилученні та валідації даних
18. Приклади перевірки на валідність вхідних даних у модулі View
19. Дослідження режимів обмеження ON DELETE
20. Ілюстрації програмного коду на Github

# Лістинги програми з директивами внесення, редагування та вилучення даних у базі даних та результати виконання цих директив

## **Функції для внесення**

Внесення відбувається через метод insert, який приймає назву таблиці, колонки до яких додаються дані, дані, і кількість рядків. Якщо на вхід подається масив пар, то при значенні bool = true, вони інтерпретуються як функції SQL.

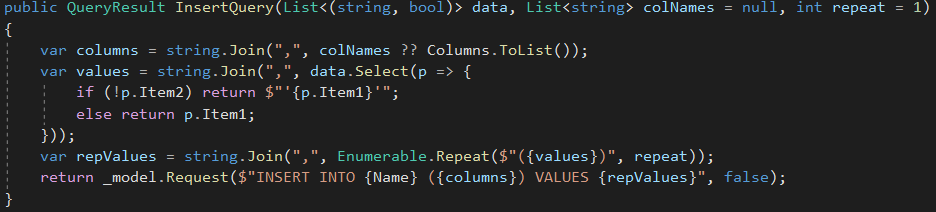


Рис.10 – функція InsertQuery

Класс NewsFeedView спадковує клас View і реалізовує функції для зручного вводу даних, наприклад:

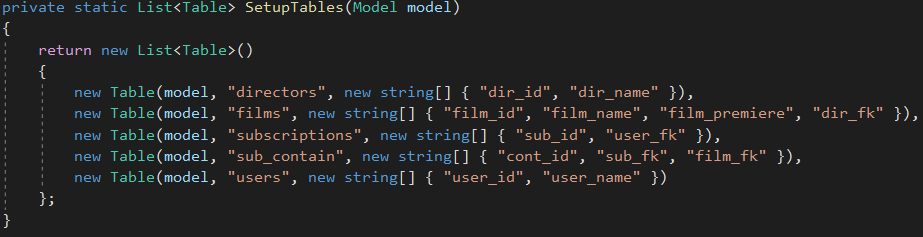


Рис.11 – функція формування таблиць БД

Введення параметрів для, вище зазначеного методу Insert у модулі ViewIO виглядає наступним чином:

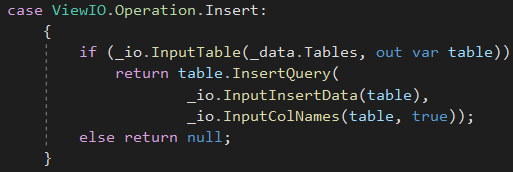
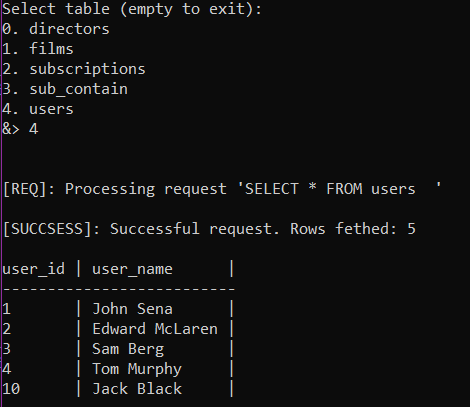


Рис.12 – функція введення даних для Insert

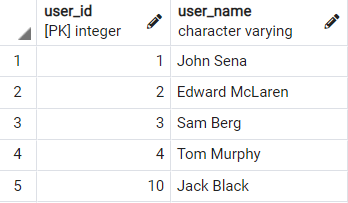
### **Приклад роботи**



Результат роботи, використовуючи оператор виведення всіх полів:



Результат роботи, використовуючи інтерфейс pgAdmin:



Аналогічним чином працює і для інших таблиць бази данних.

## **Функції для оновлення**

Оновлення відбувається через функцію UpdateQuery, якиа приймає назву таблиці, колонки до яких додаються дані, дані, і умова оновлення. Аналогічно до функції insert якщо на вхід подається масив пар, то при значенні bool = true, вони інтерпретуються як функції SQL..

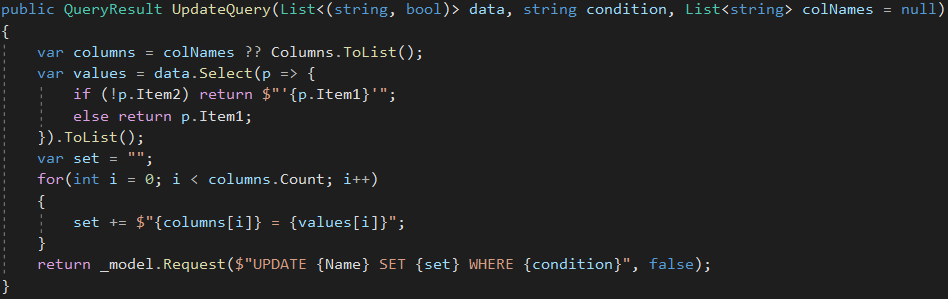


Рис.13 – функція UpdateQuery

Виклик, аналогічний до функції insert, у модулі View:

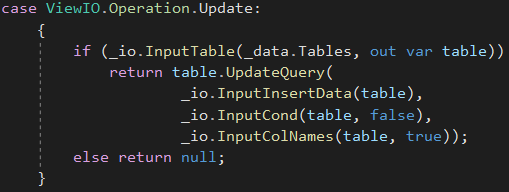
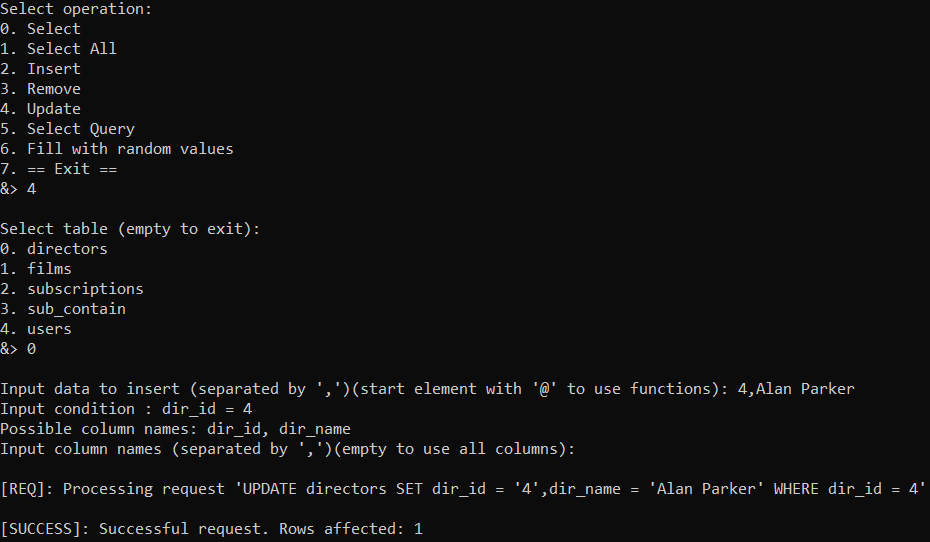
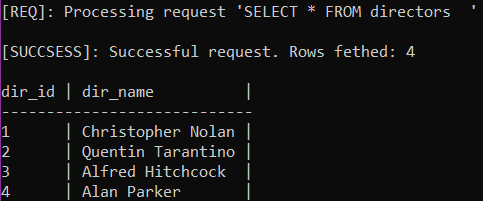


Рис.14 - функція введення даних для Update

### **Приклад роботи**



Результат роботи, використовуючи оператор виведення всіх полів

Результат роботи, використовуючи інтерфейс pgAdmin:

### 

### **Функції для вилучення**

Вилучення відбувається через функцію delete, який приймає назву таблиці і умову видалення:

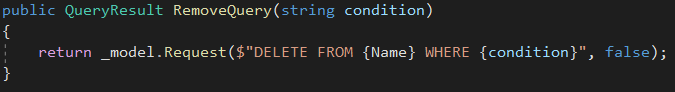
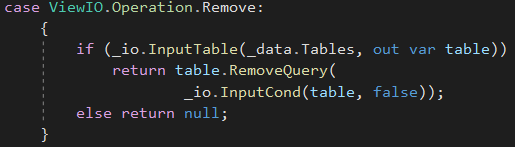
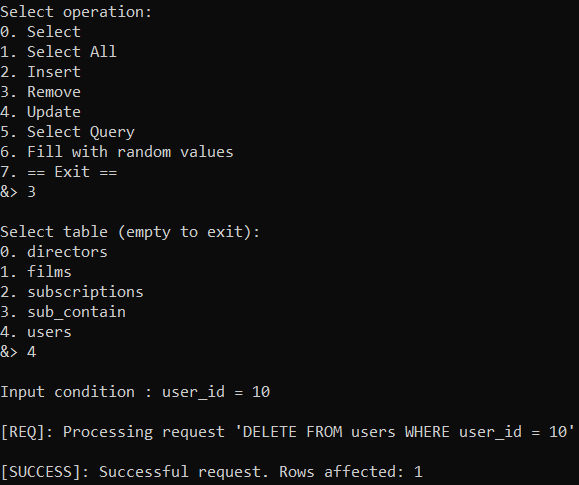


Рис.15 – функція видалення RemoveQuery

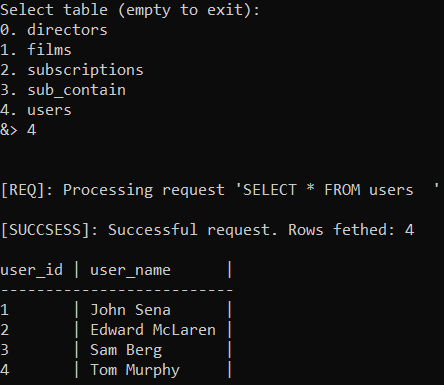
Виклик фунції у модулі View:



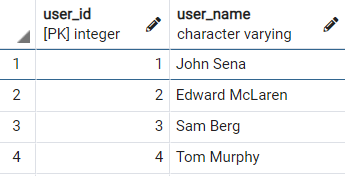
### **Приклад виклику**



### **Результат роботи**



Результати роботи в pgAdmin



# Лістинги програми з директивами внесення рандомізованих даних і виконання динамічних запитів у базі даних та результати виконання цих директив

## *Рандомізоване внесення даних до таблиці «Directors»*

Метод визначений у контролері, та викликається модулем Program. Він викликає, вже розглянуту функцію insert, заповняючи поля функцією SQL:

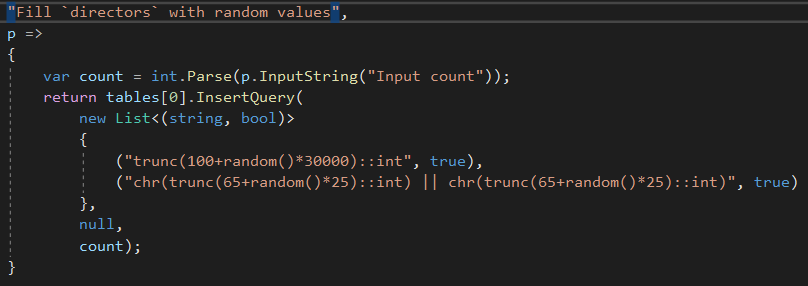
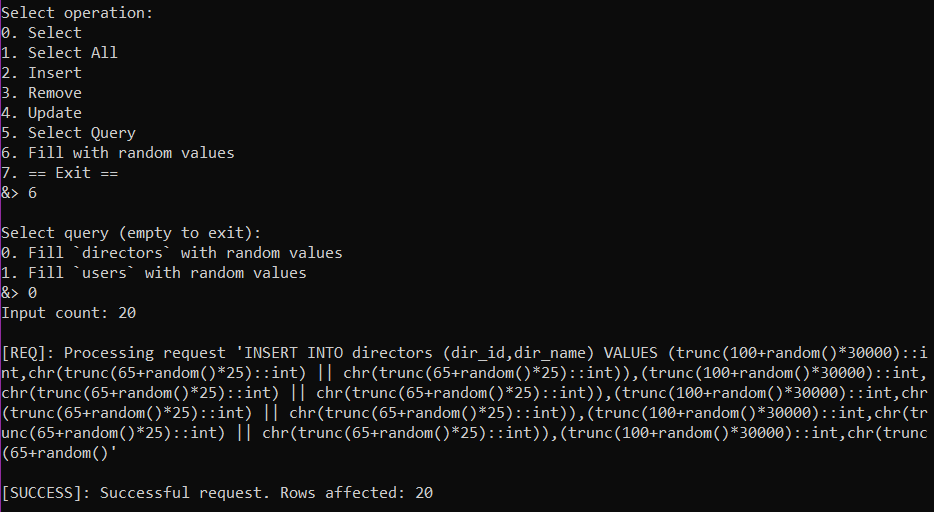
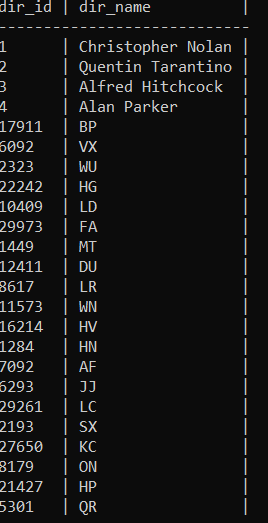


Рис.16 – функція заповнення рандомізованих даних

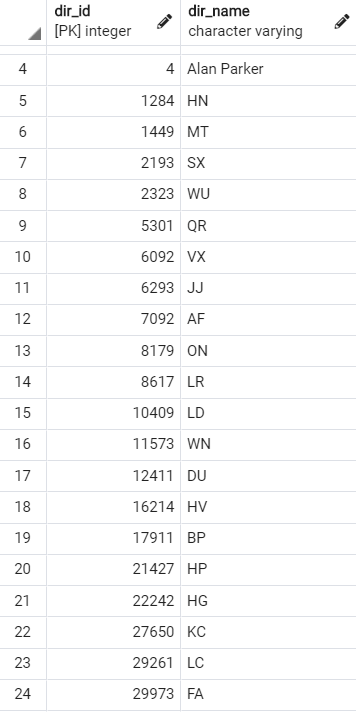
### **Приклад використання**



### **Результат виконання**



Результати роботи в pgAdmin



# Виконання динамічних запитів бази даних

Можливий масив запитів перевизначений в контролері:

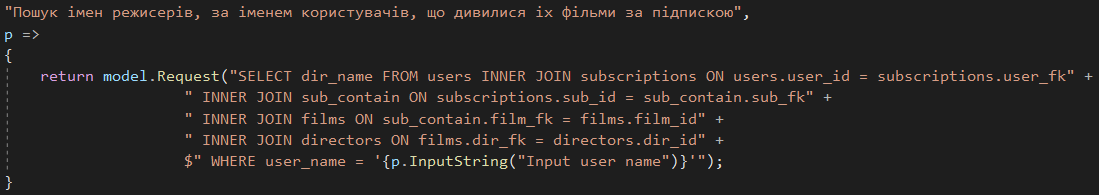


Рис.17 – функція динамічного запиту

Для виконання динамічних запитів використовується функція InputString, яка приймає список таблиць які будуть об’єднані, список атрибутів, які необхідно отримати, список атрибутів, по яким відбуватиметься зв’язування (може буди кортеж, якщо зв’зування з наступним відбуватиметься з іншим ключем, ніж з попереднім), та список обмежень пошуку

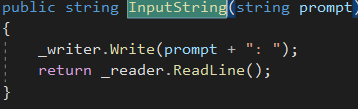
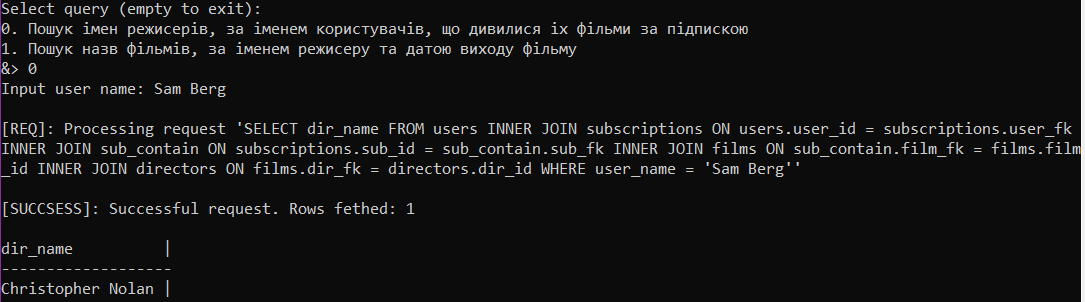


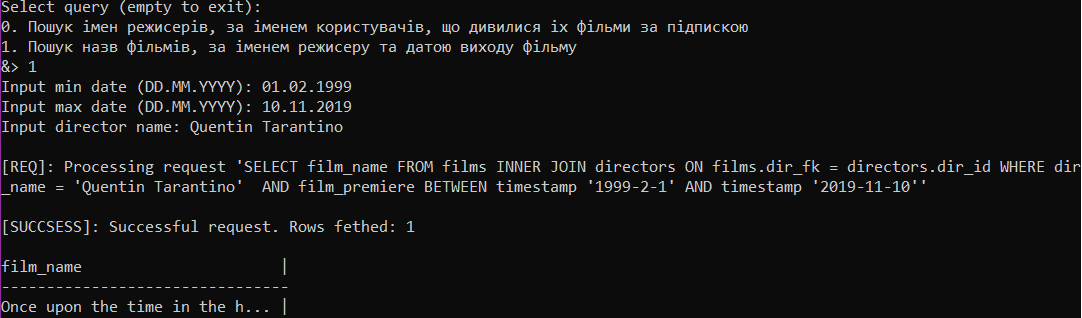
Рис.18 – функція InputString

**Приклад виконання запитів**

## *Перший запит*

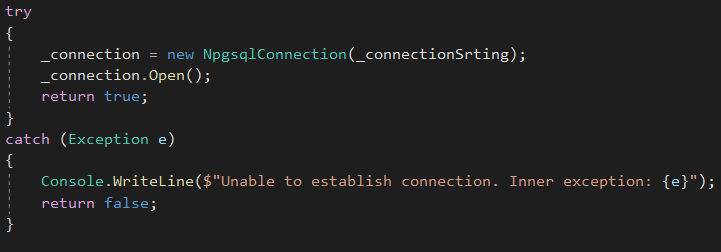


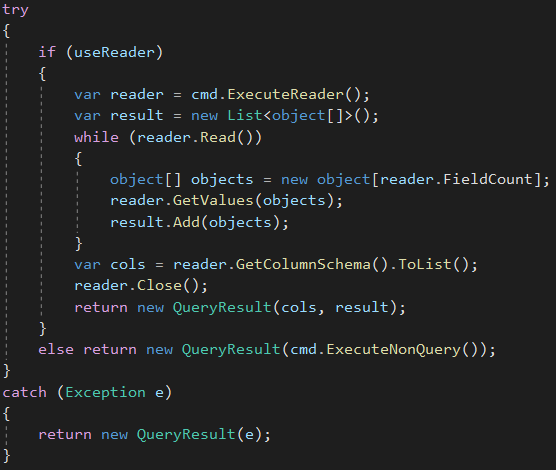
## *Другий запит*



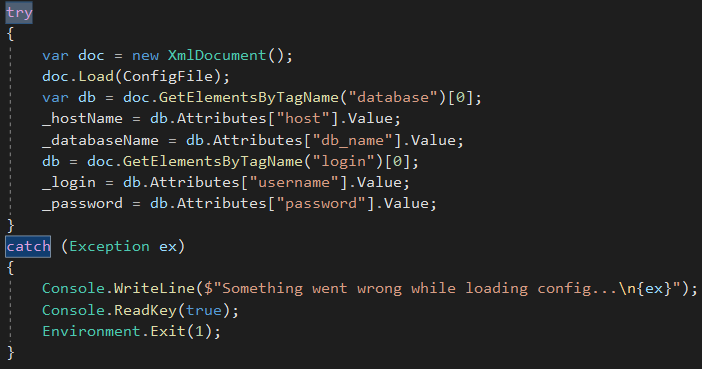
# Обробка виняткових ситуацій (помилок) при введенні/вилученні та валідації даних

Обробка виняткових при виконанні будь-якого запиту виконується за допомогою блоку try-catch у модулі Model. При введенні помилкових даних помилка серверу SQL не зупинить роботу програми, а в меню користувача з’явиться повідомлення про помилку.





## Перевірки на валідність вхідних даних у модулі Program





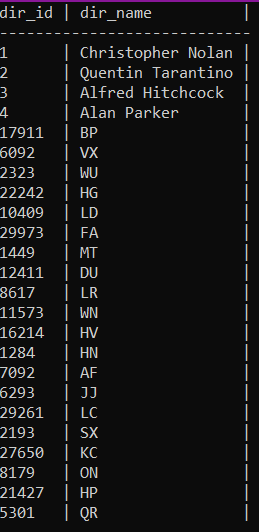
# Приклади виконання запитів

# 

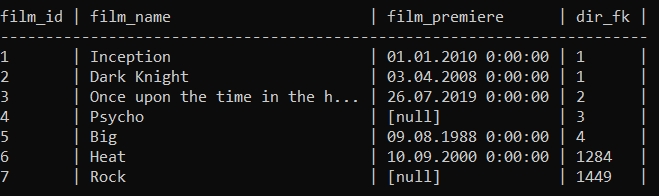
# 

# Дослідження режимів обмеження ON DELETE

*Початкові значення таблиці Directors:*

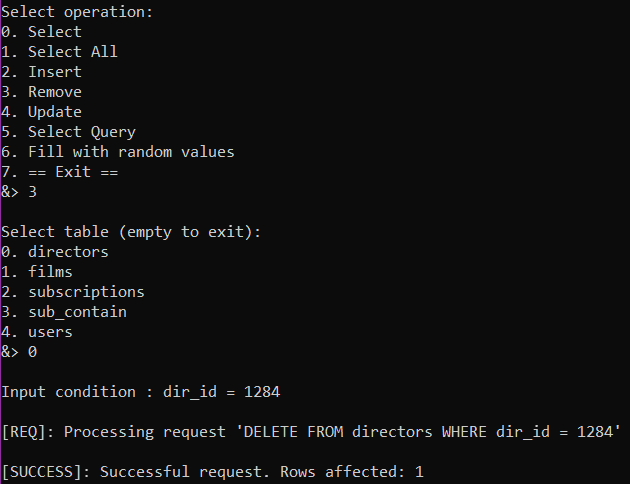


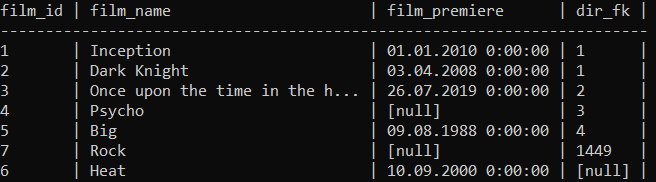
*Початкові значення таблиці Films:*



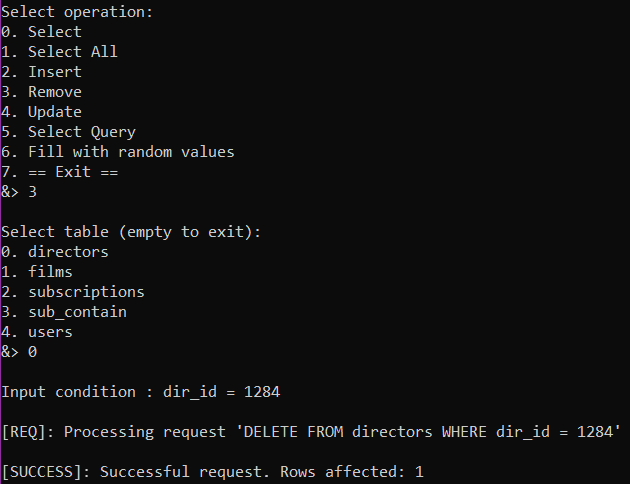
Режим **NO ACTION**, **SET NULL** (за умови що user\_key має обмеження NOT NULL), **SET DEFAULT**, **RESTRICT**

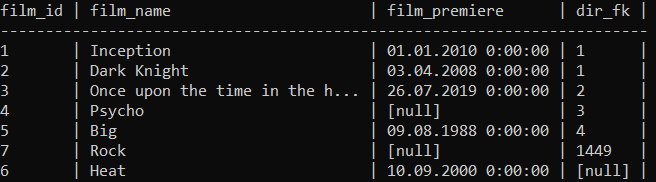
*Режим SET NULL*



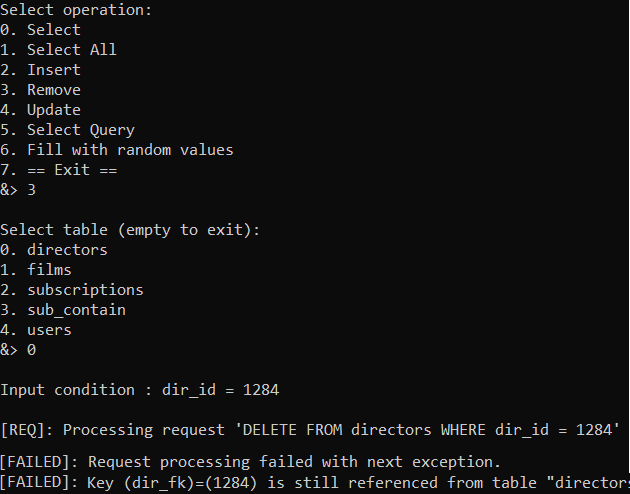


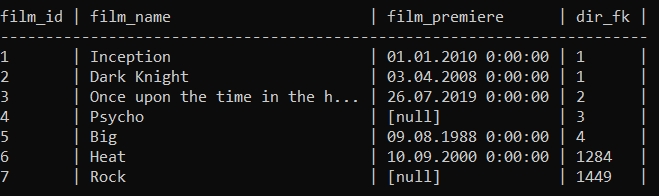
*Режим SET DEFAULT*



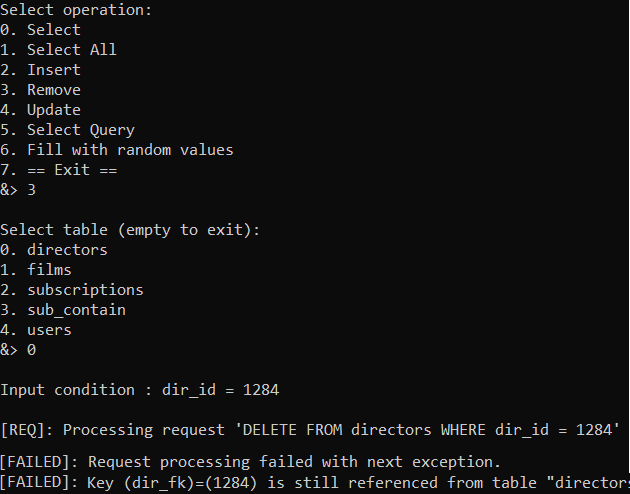


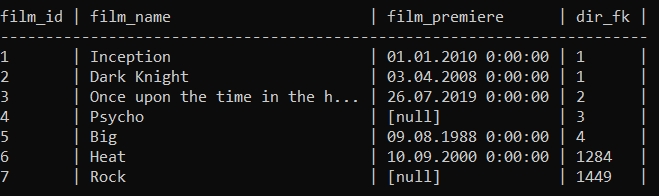
*Режим NO ACTION*



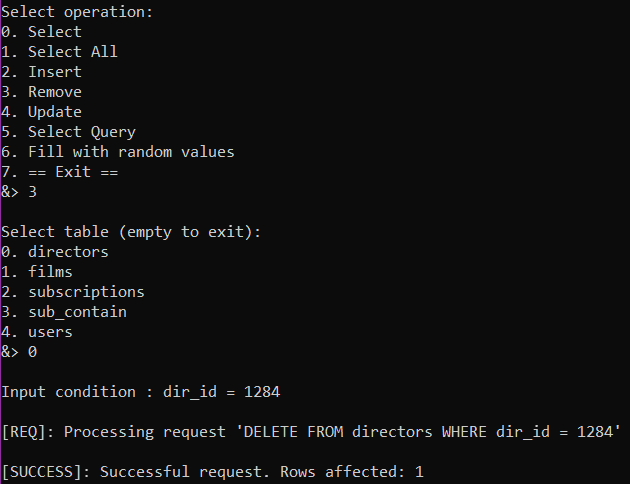


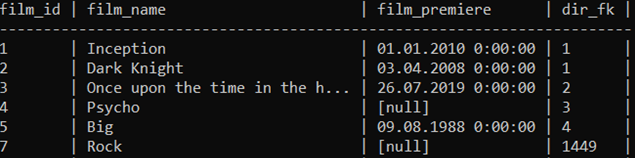
*Режим RESTRICT*





*Режим CASCADE*





# Ілюстрації звіту на Github